Embedded Systems Programming

Interrupt Processing in Linux (Module 14)

Yann-Hang Lee
Arizona State University
yhlee@asu.edu
(480) 727-7507

Summer 2014

Example of I2C Devices

- **Two Wii nunchuck devices**
  - one is connected to ICH8 SMBus
  - one is connected to I2C adapter on PCI bus
- **2 instances of I2C_client**
  - different I2C device names
  - different adapters
  - Use the same device driver
  - same I2C slave address 0x52
- **When read from the nunchucks**
  - Same I2C signals on both buses, e.g. start, addr, R/W, ack …..
  - Different commands are sent to the different adapters (ICH8 SMBus module and PCI I2C adapter)
  - Driver makes the same call to

    i2c_smbus_xfer of *i2c.core* and then *adapter->algo->smbus_xfer*

User Space Access to I2C Devices

- **Basically, a device driver to control I2C adapters**
  - Send and receive raw data to and from I2C buses
- **An I2C device driver can process the raw data and present data according to device model**
  - A nunchuck device driver measures the speed of joystick movement instead of reporting joystick position.
- **I2C-dev – loadable module**
  - Major number: 89
  - Minor number: defined for each adapter
  - i2c_dev represents an i2c_adapter, an I2C or SMBus master, not a slave (i2c_client) – called */dev/i2c-0, /dev/i2c-1, /dev/i2c-2,* etc.

```
struct i2c_dev {
        struct list_head    list;
        struct i2c_adapter    *adap;
        struct device    *dev;          };
```

How to Use I2C-dev

- **install i2c-dev module**
- **Examine the device created (“i2c-0”)**
- **Include i2c-dev.h where i2c-dev interface is defined**

```
#define ADDRESS 0x54
int fd;
fd = open( "/dev/i2c-0", O_RDWR );          // open a device file
ioctl( fd, I2C_SLAVE, ADDRESS );            // set up the slave address
```

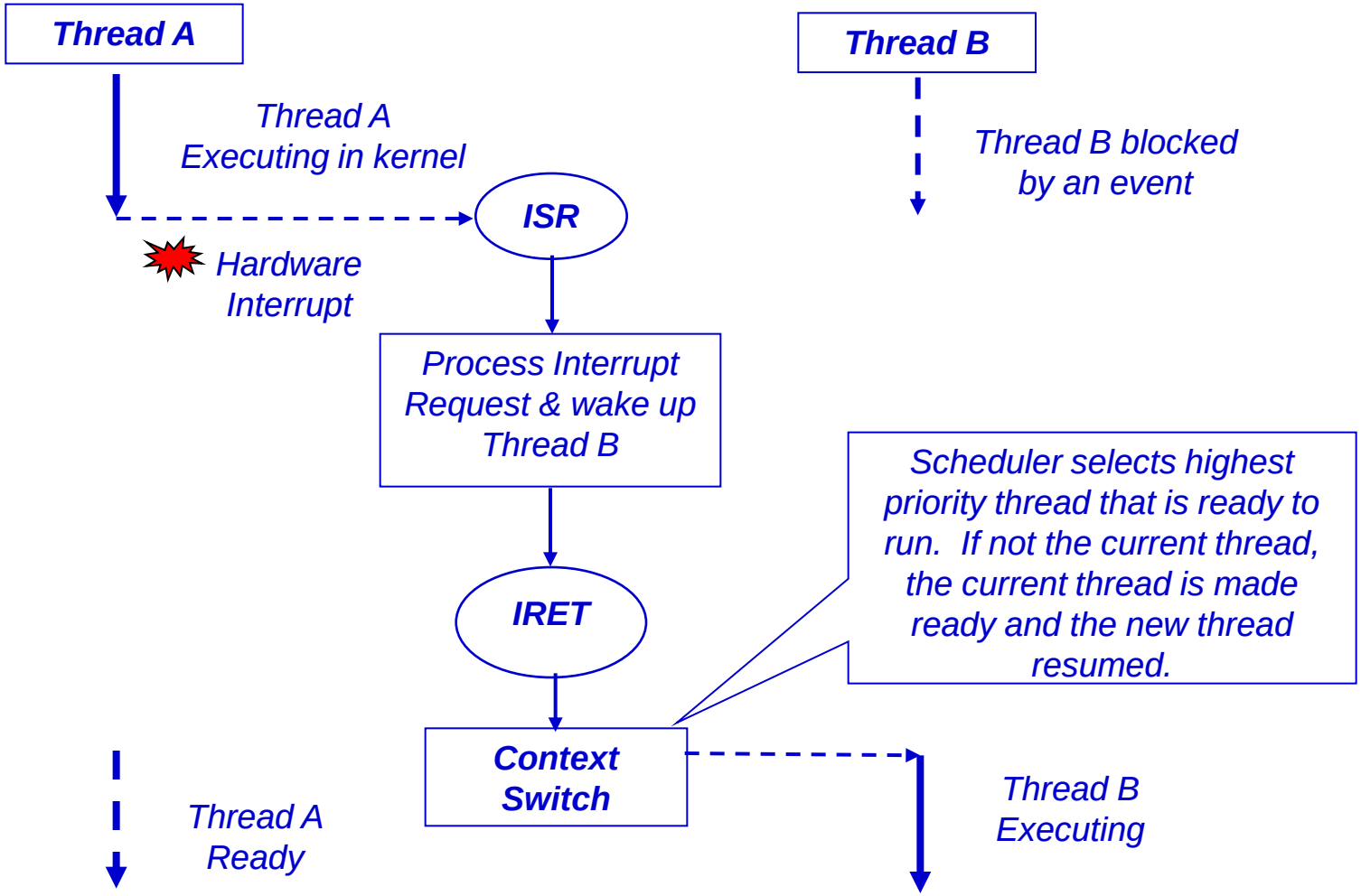
- Using read() and write() for an entire I2C transaction takes place (i.e. start bit, address, data, stop).
- Using the wrapper functions that i2c-dev.h provides.
- SMBus commands

  i2c_smbus_write_byte_data() → *i2c_smbus_acces*

  → *ioctl(file,I2C_SMBUS,&args)*

  → *S _ Addr _ Wr _ [A] _ Comm _ [A] _ Data _ [A] _ P*

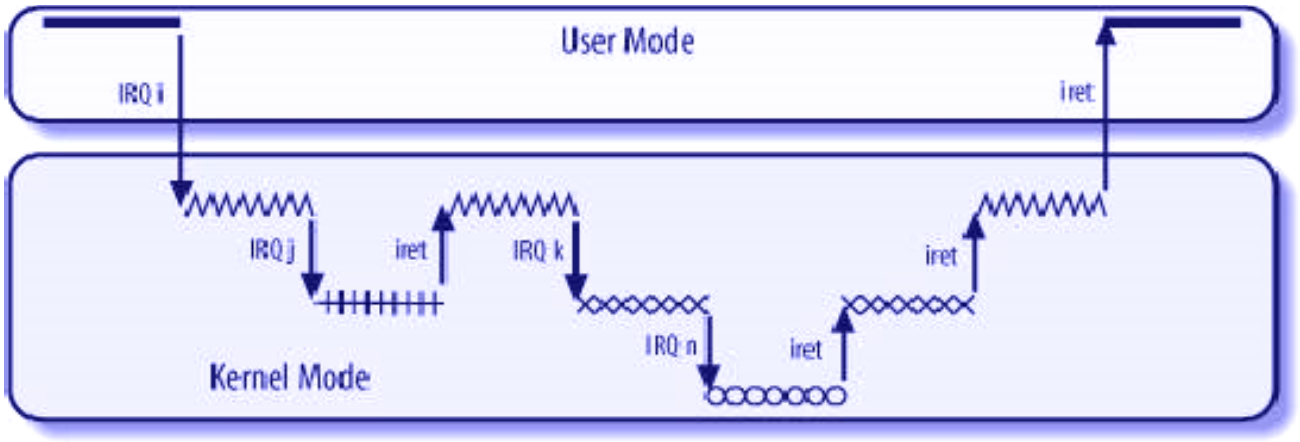
Interrupt and Preemptive Context Switching



(ftp://ftp.prenhall.com/pub/esm/electrical_engineering.s-037/lewis/powerpoint/chapter_7.zip)

Nested Execution of Handlers

- **Generally nesting of kernel code paths is allowed with certain restrictions**
- **Exceptions can nest only 2 levels**
  - Original exception and possible Page Fault
  - Exception code can block
- **Interrupts can nest arbitrarily deep, but the code can never block (nor should it ever take an exception)**



(D. P. Bovet and M. Cesati, "Understanding the Linux Kernel", 3rd Edition)

Interrupt Handling

- **Depends on the type of interrupts**
  - *I/O interrupts*
  - *Timer interrupts*
  - *Interprocessor interrupts*
- **Unlike exceptions, interrupts are “out of context” events**
- **Generally associated with a specific device that delivers a signal on a specific IRQ**
  - IRQs can be shared and several ISRs may be registered for a single IRQ
- **ISRs is unable to sleep, or block**
  - *Critical:* to be executed within the ISR immediately, with maskable interrupts disabled
  - *Noncritical:* should be finished quickly, so they are executed by theISR immediately, with the interrupts enabled
  - *Noncritical deferrable:* deferrable actions are performed by means of separate functions

I/O Interrupt Handling